Quando trabalhamos com NodeJS, é comum usarmos arquivos diferentes para separar e organizar o código. Cada arquivo `.js` é um módulo independente e suas funções, variáveis e classes não são compartilhados com o restante do código, a não ser quando são **explicitamente exportados e importados em outros módulos**.

O JavaScript, em seus diversos interpretadores, faz a importação/importação de módulos de duas formas, usando a sintaxe CommonJS ou CJS ou a sintaxe EcmaScript Modules, ou ESM. Vamos ver exemplos de ambas as formas.

CommonJS, ou CJS

Até a versão 13, a função que o NodeJS utiliza por padrão para importar módulos em um arquivo é `require()`. Os módulos podem importar e exportar todas as funções declaradas no arquivo ou apenas algumas, de acordo com o necessário. **Este é o formato de exportação e importação de módulos conhecido como CommonJS ou CJS**.

Por exemplo, para exportar apenas uma função em um arquivo:

```
module.exports = function soma(num1, num2) {
 return num1 + num2;
};
```

ou

```
function soma(num1, num2) {
 return num1 + num2;
}

module.exports = soma;
```

É possível exportar apenas a função que precisa ser executada a partir de outra parte do código, enquanto outras funções do mesmo arquivo permanecem inacessíveis ao restante. Por exemplo:

```
function soma(num1, num2) {
 return num1 + num2;
}

function multiplica(num1, num2) {
 return soma(num1, num2) * 2;
}

module.exports = multiplica;
```

No exemplo anterior, declaramos as funções `soma()` e `multiplica()`. A função `multiplica()` executa `soma()` dentro de seu escopo e retorna um resultado. Podemos exportar somente a função `multiplica()` para ser utilizada em outras partes do código, sem a necessidade de exportar tudo.

Por outro lado, caso seja necessário exportar e importar mais de uma função, podemos fazer isso exportando um único objeto no final do arquivo/módulo:

```
function soma(num1, num2) {
 return num1 + num2;
}

function multiplica(num1, num2) {
 return soma(num1, num2) * 2;
}

module.exports = { multiplica, soma };
```

Essas funções podem ser importadas desestruturando o mesmo objeto:

```
const { multiplica, soma } = require('./caminho/arquivo');

const resultadoMult = multiplica(2, 2);
const resultadoSoma = soma(2, 2);
```

O mesmo princípio se aplica para módulos externos que instalamos em nosso projeto a partir de um gerenciador de pacotes como o NPM (com `npm install <nome da lib>`); nesse caso não precisamos passar o caminho, pois o JavaScript vai acessar os métodos necessários a partir da pasta `node_modules`:

```
const chalk = require('chalk');
```

A mesma coisa para módulos que já são *built-in* no NodeJS. Ou seja, são módulos que integram o sistema, mas mesmo assim precisam ser importados para dentro do projeto, como o módulo `fs` (file system, ou sistema de arquivos):

```
const fs = require('fs');
```

Já vimos como o `chalk` funciona e veremos o `fs` mais para frente no curso.

EcmaScript Modules, ou ESM

Quando utilizamos o ESM, o mesmo processo de exportação de módulos é feito com a sintaxe `export` ou `export default` e a

importação com a sintaxe `import <nomeModulo> from './caminho/arquivo.js'`.
Esta outra forma de lidar com a importação e exportação de módulos veio com o famoso ES6 ou JS2015 e foi aos poucos sendo implementada para funcionar nativamente nos navegadores com a ajuda de *bundlers* como o WebPack, que fazem a "tradução" de métodos do JavaScript mais moderno para garantir retrocompatibilidade.

Hoje o ESM já funciona nativamente em todos os navegadores e passou a ter suporte para Node a partir da versão 13 (no momento em que escrevemos este conteúdo, o NodeJS está na versão 16.13.2). Mesmo assim, grande parte das aplicações desenvolvidas com NodeJS ainda utiliza o formato CJS de importação e exportação de módulos e as bibliotecas, pacotes e frameworks estão em processo de substituição do CJS para o ESM.

A sintaxe do ESM segue os exemplos abaixo. Para exportar funções, por exemplo:

```
export function soma(num1, num2) {
 return num1 + num2;
}

export function multiplica(num1, num2) {
 return soma(num1, num2) * 2;
}
```

Ou, alternativamente, para exportar somente a função `multiplica()` como *default* (padrão):

```
function soma(num1, num2) {
 return num1 + num2;
}

function multiplica(num1, num2) {
 return soma(num1, num2) * 2;
}

export default multiplica;
```

Ou outra forma de exportar mais de uma função:

```
function soma(num1, num2) {
 return num1 + num2;
}

function multiplica(num1, num2) {
 return soma(num1, num2) * 2;
}
```

```
export { multiplica, soma };
```

Para fazer a importação, seguem os exemplos:

```
import soma from './caminho/arquivo.js';
```

Ou, no caso de mais funções exportadas a partir do mesmo módulo:

```
import { soma, multiplica } from './caminho/arquivo.js';
```

É possível também importar de uma só vez todo o objeto exportado:

```
import * as operacoes from './caminho/arquivo.js';
```

E importar da seguinte forma:

```
const soma = operacoes.soma(1, 2);
const multiplica = operacoes.multiplica(1, 2);
```

Importante: para utilizar a sintaxe ESM com NodeJS é preciso incluir, no arquivo `package.json`, a propriedade `"type": "module"` e sempre incluir a extensão do arquivo `.js` nos caminhos de importação - por exemplo `import soma from './caminho/arquivo.js';`

Existe uma convenção no uso de ESM em projetos NodejS, que é utilizar a extensão `.mjs` para distinguir quais arquivos são módulos, continuando com a extensão `.js` para os arquivos que não exportam módulos.

Conclusão

- O sistema CJS (CommonJS) foi desenvolvido para funcionar como o sistema de exportação/importação de módulos do NodeJS.
- O ESM (EcmaScript Modules) foi desenvolvido para que o JavaScript tivesse nativamente seu próprio sistema de módulos - estamos falando do JavaScript interpretado nos navegadores.
- O NodeJS implementou o suporte ao ESM a partir da versão 13.